EXHIBIT D

5.0 Reference

This section describes the actual OLE Interfaces exposed, the definitions of the data structures used when passing data around, and the definitions of each class used internally by the driver.

5.1 Interfaces

The following diagram describes all interfaces exposed by the driver specific to driver-component interpretability. For a complete list of all XMCSPI interfaces see the *XMC SPI Reference Guide.*

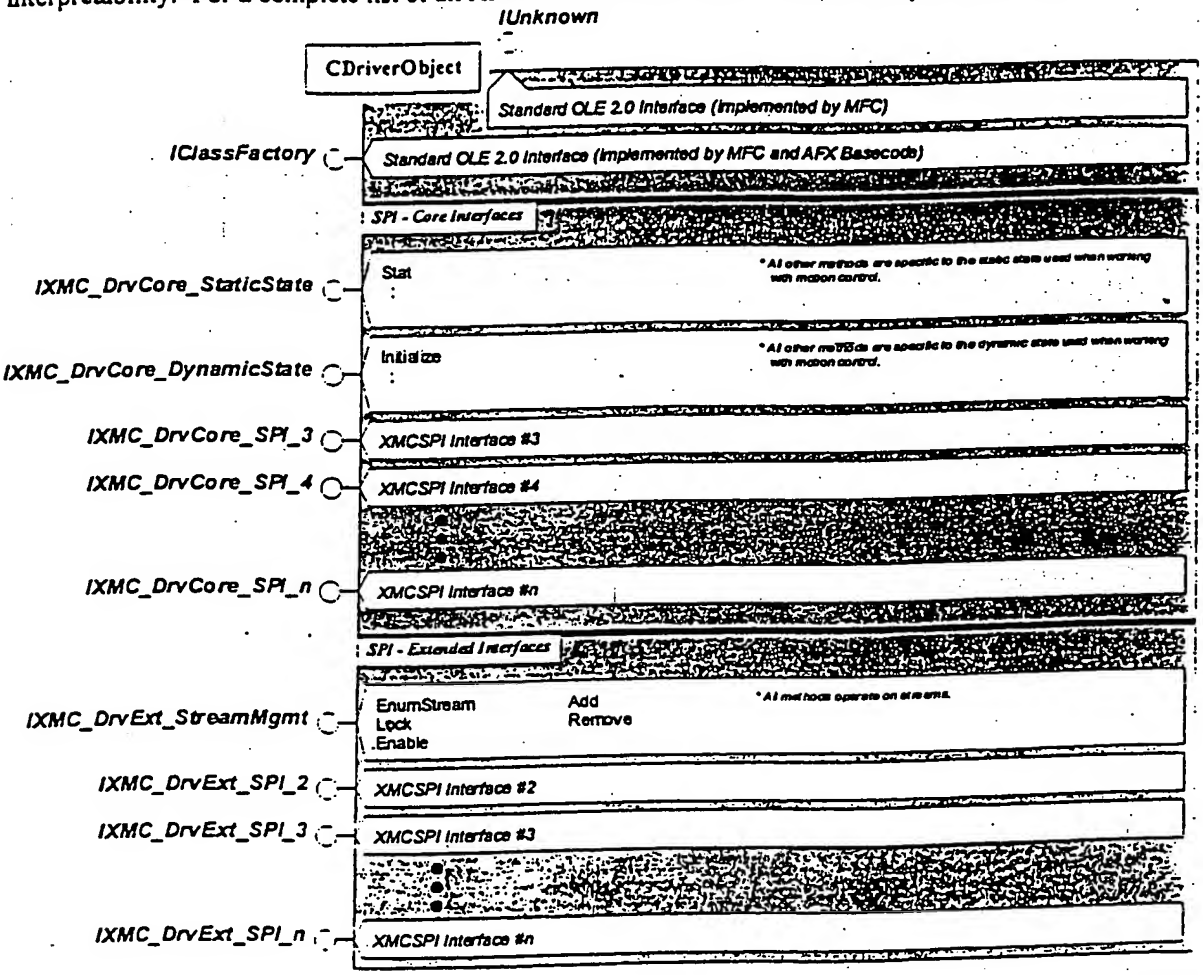

Figure 11 *Interface-Map.*

Other than the two standard interfaces exposed, **IUnknown** and **IClassFactory**, there are three other interface used when either the component or driver administrator communicate with the driver. These interfaces are both the **IXMC_DrvCore_StaticState**, **IXMC_DrvCore_DynamicState**, and **IXMC_DrvExt_StreamMgmt** interfaces. All other interfaces are XMCSPI specific and are used for the sole purpose of performing motion control operations. The following sections describe the pertinent methods in each of the key interfaces used when linking the component, driver, and stream together.

5.1.1 IXMC_DrvCore_StaticState Interface

The following exposed methods in the IXMC_DrvCore_StaticState interface are used when linking the component, driver, and stream components.

```
IXMC_DrvCore_StaticState Interface
{
        HRESULT Stat( LPXMC_DRIVER_INFO lpDI );
        :
};
```

5.1.2 IXMC_DrvCore_DynamicState Interface

The following exposed methods in the IXMC_DrvCore_DynamicState interface are used when linking the component, driver, and stream components.

```
IXMC_DrvCore_DynamicState Interface
{
        HRESULT Initialize( LPXMC_DRIVER_INFO lpDI );
        :
};
```

5.1.3 IXMC_DrvExt_StreamMgmt Interface

The following exposed methods in the IXMC_DrvExt_StreamMgmt interface are used when linking the component, driver, and stream components.

```
IXMC_DrvExt_StreamMgmt Interface
{
        HRESULT EnumStream( LPXMCENUMSTREAM lpES );
        HRESULT Lock( BOOL bLock );
        HRESULT Enable( BOOL bEnable );
        HRESULT Add( LPCLSID lpStreamCLSID, BOOL bEnable );
        HRESULT Remove( LPCLSID lpStreamCLSID );
};
```

5.2 Exported Functions

The following are the functions exported by the driver DLL.

```
XMC_DRIVER_MODULETYPE    DLLGetModuleType( void );
LPCLSID                  DLLGetCLSID( void );
BOOL                     DLLRegisterServer( void );
BOOL                     DLLUnRegisterServer( void );
```

5.3 Structures and Defines

This section defines all structures, enumerations, and defines used by the driver.

5.3.1 XMC_DRIVER_MODULETYPE Enumeration

This enumeration defines the type of drivers available. Each driver must return its type when the user calls the exported **DLLGetModuleType** function.

```
enum XMC_DRIVER_MODULETYPE
{
        XMC_DRIVER_MT           = 0x4000,
        XMC_DRIVER_MT_AT6400    = 0x4001,
        XMC_DRIVER_MT_DMC1000   = 0x4002,
        XMC_DRIVER_MT_DT2000    = 0x4003,
        XMC_DRIVER_MT_CUSTOM    = 0x4004
};
```

5.3.2 XMC_DRVCORE_CMD Enumeration

The **XMC_DRVCORE_CMD** enumeration defines an identifier for every command known to the XMC Driver. For example, every core XMCSPI function has a corresponding **XMC_DRVCORE_CMD** identifier. This index is used to look up the string block for the command. The definition of the enumeration is as follows.

```
enum XMC_DRVCORE_CMD
{
        XMC_DCC_MOTION_MOVEABS,
        XMC_DCC_MOTION_KILL,
        :
};
```

5.3.3 XMC_DRVEXT_CMD Enumeration

The **XMC_DRVEXT_CMD** enumeration defines an identifier for every extended command known to the XMC Driver. Even though the identifiers exist, the driver may or may not implement the set of commands. For example, every extended XMCSPI function has a corresponding **XMC_DRVEXT_CMD** identifier. This index is used to look up the string block for the command (if the driver implements the command). The definition of the enumeration is as follows.

```
enum XMC_DRVEXT_CMD
{
        XMC_DCE_MOTION_MOVEREL,
        :
};
```

5.3.4 XMC_DATATYPE Enumeration

The **XMC_DATATYPE** enumeration defines all types of data blocks that may be parsed from response strings returned by stream targets.

```
enum XMC_DATATYPE
{
        XMC_DT_DOUBLENUMBLOCK,
        XMC_DT_DWORDNUMBLOCK,
        XMC_DT_STRINGBLOCK
};
```

5.3.5 XMC_DATABLOCK Structure

The **XMC_DATABLOCK** structure stores all types of data that may be parsed from response strings returned by stream targets.

```
struct XMC_DATABLOCK
{
        DWORD  m_dwDataCount;

        union {
                LPDOUBLE rgDouble;
                LPDWORD  rgDWORD;
                LPTSTR   pszText;
        };
};
```

5.3.6 XMC_DRIVER_INFO Structure

The following structure is used when setting up and querying the state of the driver.

```
struct XMC_DRIVER_INFO
{
        XMC_HDRIVER                 [in,out]    m_hDriver;
        XMC_DRIVER_MODULETYPE       [out]       m_mt;
        BOOL                        [out]       m_bStreamMgmtLocked;
        LPTSTR                      [out]       m_pszHardwareVendor;
        DWORD                       [in]        m_cbMaxHardwareVendor;
        LPTSTR                      [out]       m_pszHardwareModel;
        DWORD                       [in]        m_cbMaxHardwareModel;
        LPTSTR                      [out]       m_pszDriverVendor;
        DWORD                       [in]        m_cbMaxDriverVendor;
};
```

5.4 Classes

This section contains the definition of all classes used by the driver component in its implementation.

5.4.1 CDriverDisp Class

The **CDriverDisp** class acts as the dispatch, or control, object used to separate all OLE details from the code implementing the driver internals. This object coordinates all operations taking place in the driver by directing other C++ object to carry out the work of the operation. The following is the definition of the **CDriverDisp** class.

```
class CDriverDisp
{
    public:
        //---- Constructors & Destructors ----

        CDriverDisp( void );
        ~CDriverDisp( void );

        //---- IXMC_DrvCore_StaticState Methods ----

        DWORD Stat( LPXMCDRIVERINFO lpDI );
        :

        //---- IXMC_DrvCore_DynamicState Methods ----

        DWORD Initialize( LPXMCDRIVERINFO lpDI );
        :

        //---- IXMC_DrvExt_StreamMgmt Methods ----

        DWORD Lock( BOOL bLock );
        DWORD Enable( BOOL bEnable );
        DWORD Add( LPCLSID lpStreamCLSID, BOOL bEnable );
        DWORD Remove( LPCLSID lpStreamCLSID );


        //---- IXMC_SPI Interfaces ----
        :

    private:
        //---- Private Data ----

        CCommandMgr   m_cmdMgr;
        CStreamMgr    m_strmMgr;
};
```

5.4.2 CStreamMgr Class

The CStreamMgr class is used to manage all stream operations. For example, this class is used to add new streams, remove existing streams, and send commands to all enabled streams. The definition of the CStreamMgr class is as follows.

```
class CStreamMgr
{
        public:
                //---- Constructors & Destructors ----

                CStreamMgr( void );
                ~CStreamMgr( void );


                //---- Initialization ----

                DWORD Initialize( void );


                //---- Query Operations ----

                DWORD GetStreamCount( LPDWORD pdwCount, BOOL bEnabledOnly );
                DWORD GetStreamAtIdx( DWORD dwIdx, LPSTREAM pStrm );


                //---- Passive Actions ----

                DWORD AddStream( LPCLSID lpStreamCLSID, BOOL bEnable );
                DWORD RemoveStream( LPCLSID lpStreamCLSID );
                DWORD EnableStream( LPCLSID lpStreamCLSID, BOOL bEnable);


                //---- Active Actions ----

                DWORD FireCommand( LPCOMMAND pCmd, LPRESPONSE pRsp );


        private:
                //---- Private Data ----

                CCommandMgr   m_cmdMgr;
                CStreamMgr    m_strmMgr;
};
```

5.4.3 CStream Class

The CStream class is used to directly control the stream component. All OLE details corresponding to controlling the stream component are encapsulated with in this class. The following is the definition of the CStream class.

```
class CStream
{
        public:
                //---- Constructors & Destructors ----

                CStream( void );
                ~CStream( void );


                //---- Initialization ----

                DWORD Initialize( void );
                DWORD Open( void );
                DWORD Close( void );
```

```
            //---- Setup Operations ----

            DWORD Setup( LPSTREAM_INFO lpSI );
            DWORD Stat( LPSTREAM_INFO lpSI );
            DWORD Enable( BOOL bEnable );


            //---- Connection to the stream ----

            DWORD Attatch( LPCLSID lpStreamCLSID, BOOL bEnable );
            DWORD Attatch( LPUNKNOWN lpStreamUnk, BOOL bEnable );
            DWORD Detach( LPUNKNOWN lpStreamUnk );


            //---- Actions ----

            DWORD Write( LPTSTR pszData, DWORD dwSize );
            DWORD Read( LPTSTR pszBuf, DWORD dwMax, LPDWORD lpdwRead );


      private:
            //---- Private Data ----

            LPXMCSTREAMINIT     m_lpifStreamInit;
            LPXMCSTREAM         m_lpifStream;
            BOOL                m_bEnabled;
            CLSID               m_clsid;
);
```

5.4.4 CCommandMgr Class

The **CCommandMgr** class is used to coordinate the process of building the getting the **CCommand**, building the **CResponse**, and sending both to the **CStreamMgr** for processing. The following is the definition of the **CCommandMgr** class.

```
class CCommandMgr
{
      public:
            //---- Constructors & Destructors ----

            CCommandMgr( void );
            ~CCommandMgr( void );


            //---- Initialization ----

            DWORD Initialize( void );
            DWORD SetStreamMgr( LPSTREAMMGR lpStrmMgr );


            //---- Actions ----

            DWORD FireCoreCommand( XMC_DRVCORE_CMD cmdID,
                                   LPRESPONSE *ppRsp, ... );
            DWORD FireExtCommand( XMC_DRVEXT_CMD cmdID,
                                  LPRESPONSE *ppRsp, ... );


      private:
            //---- Private Data ----

            LPSTREAMMGR         m_pStrmMgr;
            CCommandList        m_cmdList;
);
```

5.4.5 CCommandList Class

The purpose of the **CCommandList** class is to create **CCommand** objects corresponding to the appropriate **XMC_DRVCORE_CMD** and **XMC_DRVEXT_CMD** identifiers. The following is the class definition of the **CCommandList** class.

```
class CCommandList
{
      public:
            //---- Constructors & Destructors ----

            CCommandList( void );
            ~CCommandList( void );


            //---- Initialization ----

            DWORD Initialize( BOOL bAutoLoad );
            DWORD Load( void );

            //---- Actions ----

            DWORD CreateCoreCommand( XMC_DRVCORE_CMD cmdID,
                                     LPCOMMAND *ppCmd, ... );
            DWORD CreateExtCommand( XMC_DRVEXT_CMD cmdID,
                                    LPCOMMAND *ppCmd,... );
};
```

5.4.6 CCommand Class

The **CCommand** class is used to generate the raw text command that is sent to the stream and on to the hardware or some other stream target. The **CCommand** class definition is as follows.

```
class CCommand
{
      public:
            //---- Constructors & Destructors ----

            CCommand( void );
            ~CCommand( void );


            //---- Actions ----

            DWORD Build( LPCTSTR pszFmt, ... );
            LPCTSTR Render( void );
};
```

5.4.7 CResponseList Class

Similar to the **CCommandList** class, the purpose of the **CResponseList** class is to create **CResponse** objects corresponding to the appropriate **XMC_DRVCORE_CMD** and **XMC_DRVEXT_CMD** identifiers. The following is the class definition of the **CResponseList** class.

```
class CResponseList
{
      public:
            //---- Constructors & Destructors ----

            CResponseList( void );
            ~CResponseList( void );


            //---- Initialization ----
```

```
                DWORD Initialize( BOOL bAutoLoad );
                DWORD Load( v id );

                //---- Acti ns ----

                DWORD CreateCoreResponse( XMC_DRVCORE_CMD cmdID,
                                          LPRESPONSE *ppRsp, ... );
                DWORD CreateExtResponse( XMC_DRVEXT_CMD cmdID,
                                          LPRESPONSE *ppRsp, ... );
};
```

5.4.8 CResponse Class

The CResponse class is used to parse the raw text returned by the stream target after sending it a command. The CResponse class definition is as follows.

```
class CResponse
{
        public:
                //---- Constructors & Destructors ----

                CResponse( void );
                ~CResponse( void );


                //---- Initialization ----

                DWORD Initialize( LPCTSTR pszFmt );


                //---- Actions ----

                DWORD Build( LPCTSTR pszRawText );

                DWORD Render( LPXMC_DATABLOCK lpDB, LPXMC_DATATYPE lpDT );
};
```

5.5 Registration Database Data

Each XMC Driver is responsible for managing all streams used. In order to manage each stream over time in a persistant manner the driver stores the list of registered streams and their enabled state in the Registry. The following is the organization of the stream data in the registration database.

```
XMC.Driver.AT6400.100
    |---- 0x000000001
    |         |---- HardwareVendor = "Compumotor, Parker Hannifin Corp."
    |         |---- HardwareModel  = "AT6400 Stepper System"
    |         |---- DriverVendor   = "ROY-G-BIV Corporation"
    |         |---- DriverVersion  = 1.0
    |         |---- DriverOS       = "Windows NT 3.5; Windows 95; 32-bit"
    |         |---- Enabled        = 1
    |         |
    |         |---- Streams
    |                   |---- Enabled
    |                   |         |---- nCount = 2
    |                   |         |---- (CLSID) = "PC Bus"
    |                   |         |---- (CLSID) = "Text File"
    |                   |
    |                   |---- Disabled
    |                             |---- nCount = 1
    |                             |---- (CLSID) = "Serial I/O"
    |
    |---- 0x00000002
```